Reinforcement Learning for Sinhala Named Entity Recognition

H. M. S. Anuruddha

Reinforcement Learning for Sinhala Named Entity Recognition

H. M. S. Anuruddha
Index No: 14000059

Supervisor: Dr. Ruvan Weerasinghe

January 2019

Submitted in partial fulfillment of the requirements of the
B.Sc in Computer Science Final Year Project (SCS4124)

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: H. M. S. Anuruddha

……

Signature of Candidate Date:

This is to certify that this dissertation is based on the work of

Mr. H. M. S. Anuruddha

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Principal Supervisor's Name: Dr. Ruvan Weerasinghe

……

Signature of Principal Supervisor Date:

Co-Supervisor's Name: Dr. Ajantha Athukorale

……

Signature of Co-Supervisor Date:

Abstract

Named Entity Recognition is a subtask of Natural Language Processing. In the literature various machine learning methods have been proposed to tackle this task such as supervised learning, unsupervised learning and semi supervised learning. It is established that, for supervised learning, a large number of annotated data is needed for a system to have a good generalization capability.

For Indic languages such as Sinhala it is challenging to build a Named Entity Recognition system due to its inherent features such as lack of capitalization. Even though there have been several researches conducted for Sinhala, most of them have used supervised learning methods with small quantities of training data.

The resurgence of Reinforcement Learning demonstrates its effectiveness in problems that humans solve effectively in an incremental manner over time. Even though it is hard to find research that has applied Reinforcement Learning to Natural Language Processing tasks, there have been attempts to map tasks such as Part of Speech tagging to the Reinforcement Learning paradigm.

This dissertation first casts the Sinhala Named Entity Recognition task into the Reinforcement Learning paradigm and then proposes a language specific Named Entity Recognition system that can be trained to generalize better using the current annotated data available.

Preface

This study has been done following the SP-MDP framework that was proposed by Maes and Gallinari . Section 3.3 relies on the work that has been done by Maes and Gallinari in the paper Structured prediction with reinforcement learning. This study has adopted their framework and used as a specialized version to align it with the Named Entity Recognition task.

The learning algorithm proposed in this study is a derivative work and it has been derived from the algorithm proposed by Langford and Daume in their work, Learning to Search [1] algorithm proposed. The implementation of the derived algorithm is an outcome of this research entirely.

The dataset that has been used to train on and test against, has been prepared by the Language Technology Research Laboratory of University of Colombo School of Computing.

Common open source libraries such as Tensorflow has been used to create the Neural Network. NLP open source libraries such as gensim, has been used to create word embedding models of the corpus. Apart from the open source libraries used, the code implementation for the entire system is an outcome of this study.

Acknowledgement

I would like to express my sincere gratitude to my research supervisor, Dr. A.R. Weerasinghe, senior lecturer of University of Colombo School of Computing and my co-supervisor, Dr D.A.S.Atukorale, senior lecturer of University of Colombo School of Computing and my research advisor, Mr. K. V. D. J. P. Kumarasinghe, lecturer of University of Colombo School of Computing for providing me continuous guidance and supervision throughout the research.

I would also like to extend my sincere gratitude to my examiners and evaluatos for providing feedback on my research proposal and interim evaluation to improve my study.

I also take the opportunity to thank Dr. H. E. M. H. B. Ekanayake for all the assistance provided as the final year computer science project coordinator.

It is a great pleasure for me to acknowledge the assistance and contribution of all the people who helped me to successfully complete my research.

Table of Contents

List of Figures

List of Acronyms

NLP - Natural Language Processing

RL - Reinforcement Learning

NER - Named Entity Recognition

NE - Named Entity

POS - Part Of Speech

SP - Structured Prediction

MDP - Markov Decision Process

SP-MDP - Structure Prediction Markov Decision Process

DARPA - Defense Advanced Research Projects Agency

HMM - Hidden Markov Model

AI - Artificial Intelligence

MUC - message Understanding Conference

ME - Maximum Entropy

SVM - Support Vector Machine

CRF - Conditional Random Fields

CoNLL - Conference on Natural Language Learning

FSM - Finite State Machine

NNE - Non-Named Entity

NN- Neural Network

LTRL - Language Technology Research Laboratory

UCSC - University of Colombo School of Computing

Chapter 1 - Introduction

1.1 Background to the research

Named entities have three top-level categorizations according to DARPA's Message Understanding Conference: entity names, temporal expressions, and number expressions [2]. The entity names category describes the unique identifiers of people, locations, geopolitical bodies, events, and organizations.

Some key design decisions in an NER system are proposed in [3] that cover the requirements of NER in the example sentence above Chunking and text representation, Inference and ambiguity resolution algorithms, Modeling of Non-Local dependencies, Implementation of external knowledge resources and gazetteers.

Named entities are often not simply singular words, but are chunks of text, e.g. University of Maryland Baltimore County or The Central Bank of Australia. Therefore, some chunking or parsing prediction model is required to predict whether a group of tokens belong in the same entity. Left to right decoding, Viterbi, and beam search algorithms has been employed as chunking algorithms in the literature. Further, some NER systems are comprised primarily of text parsers as in [4], [5], and [6].

Inference refers to the ability of a system to determine that a chunk is actually a named entity, or, sometimes more importantly, to determine the classification of a named entity, especially in places where there is ambiguity. For example "Washington" might refer to either a name or a location. "Galaxy" might refer to a generic noun or the professional major league soccer team. Maximum Entropy Models, Hidden Markov Models and other statistical methods are employed to perform this analysis, usually implemented as a machine-learning system, as for instance in [7], [8], and [9].

Non-local dependency models refer to the ability to identify multiple tokens that should have the same label assignment or cross-reference. It is important to note that case becomes important here—e.g. Bengfort, bengfort, and BENGFORT should all be identified as the same entity, and these would break word-level rule based systems (e.g. find all words that are capitalized). But further, even different chunks should be identified

similarly – UMBC vs. University of Maryland Baltimore County or the inclusion of titles in one location that are absent from another as in President Barack Obama vs. Obama. The non-locality of these models refers to the usage of these terms outside the scope of a sequence of tokens that is being analyzed together (usually a sentence), but rather in the entire document or corpus. The papers that address non-local dependencies include [10] and [11], but the focus of this paper will be on solutions that require external knowledge.

Names, because they uniquely identify entities, are a domain not easily captured by even the most expansive lexicons. For example, simply creating a list of the names of all companies formed in the United States would expand drastically every single year. However, external knowledge and name lexicons are required for many of the approaches and solutions, not just non-local dependency models. Therefore the construction and use of gazetteers and other resources are necessary.

For languages such as English, which are high resourced languages which has many large annotated datasets, there exist many sophisticated tools developed to classify Named Entities with high accuracy. These models cannot be used to identify Named Entities in Sinhala language due to language dependent features such as non-existence of capitalization. Due to lack of research in this field for Sinhala, there exist only few models which is able to carry out this task. And there is no publicly available annotated data set. Considering all models that exist to solve NER, models that uses Reinforcement Learning can be hardly found. This study focuses on how to use Reinforcement Learning in Named Entity Recognition.

1.2 Research Problem and Research Questions

For Natural Language Processing (NLP) it is needed resources such as annotated corpora. Since NLP research conducted on Sinhala is very minimum, available annotated data is very minimum. Manually annotating data is time consuming. Thus it is required to automate this process. Machine Learning is a method of automating this process. However

annotated data is necessary for training an accurate Machine Learning system. Lack of annotated data is a common problem for low resourced languages such as Sinhala.

The focus in this study is on Named Entity Recognition. There exist Named Entity Recognition(NER) systems for Languages such as English with good accuracies. However these cannot be directly used for Sinhala. Sinhala is a morphologically rich language. Unlike languages such as English, Sinhala has a complex and rich structure and complex words with rich morphology. Another major problem is, Sinhala lacks features such as capitalization. Thus, the need for such a system specifically designed for Sinhala is very important.

However while there exists few NER systems for Sinhala, almost all the systems have used supervised learning as the learning method. Given the recent advances of other learning methods such as semi-supervised learning and reinforcement learning, it is assumed that reinforcement learning systems learn to generalize better than supervised learning systems. Although, whether a system could achieve better performance with RL than supervised learning is debatable. Even to compare these two techniques for NER task, it is difficult to cast the NER task as an RL problem.

The main research questions of this study are,

- How to cast Named Entity Recognition as a Reinforcement Learning problem.
- Is it possible to achieve better performance with Reinforcement Learning.

1.3 Justification for the research

As the technology evolves, Natural Language Processing has become a vital factor. World Wide Web has become a place where semantic meaning is very important. In near future, machines will have to understand human languages as us humans do, to serve humans better. Thus, it is in no doubt that Natural Language Processing has a major role in technology. For machines to understand human languages and to extract useful information, systems such as Named Entity Recognition are necessary.

Language translators such as Google translator has a general machine learning model for all the languages. Meaning, they don't train a model for each language pair, as parallel data is hard to come by. Instead transfer learning techniques are used, and this leads to poor accuracy of translations for Sinhala to and from other languages. We need a language specific translator to improve accuracy. NER is a preprocessing task in Machine Translation and Information Extraction. As such, a good NER system can improve the performance of a Language Translator.

In the literature, applying RL techniques in Natural Language Processing is not very common. This study could develop an interest of using RL techniques for other Natural Language Processing tasks in other researchers since it has been a difficult task to map these two areas.

Reinforcement is currently the most active research area in Artificial Intelligence. And Natural Language Processing is another vital aspect in Artificial Intelligence. Bridging the gap between these two components of AI, potentially opens up novel research areas thus, bringing the true Artificial Intelligence closer.

1.4 Methodology

As the first step, the existing systems for solving NER was analyzed. Literature review and related theories were covered in a breadth first manner. As the second step, a basic Reinforcement Learning model was proposed. The third step included implementing, refining, and testing this model.

Figure 1.1 shows the high level diagram of the methodology followed in this study.

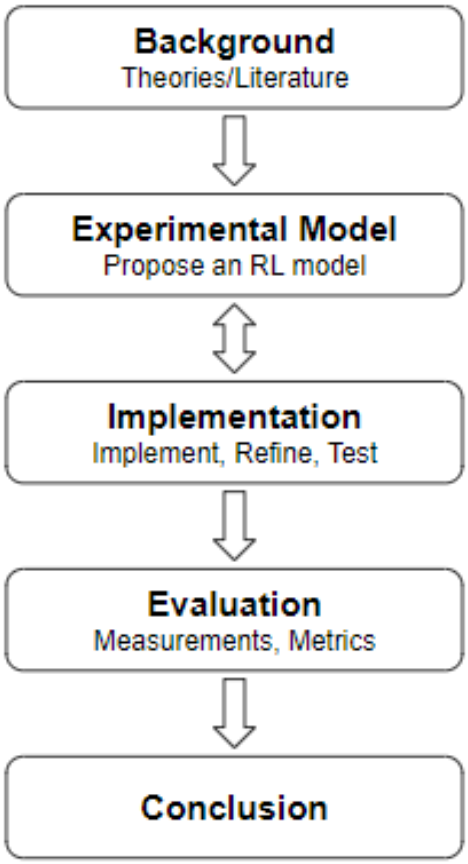

Figure 1.1 - High level diagram of the methodology followed

1.5 Outline of the Dissertation

The dissertation is structured as follows. Chapter two explores the existing approaches related to the domain of Named Entity Recognition. Chapter three describes the proposed research design and methodology. Chapter four demonstrates the implementation details of the proposed methodology. Chapter five presents the evaluation model and the evaluation results of the proposed approach. The last chapter, chapter six demonstrates the conclusion of the thesis and outlines the future work.

1.6 Delimitations of Scope

In this research, only person names, location names and organization names are considered as Named Entities. This study has focused on proposing a system that classifies each word into two classes, Named Entities and Not-Named Entities. Depending on the time availability, this study will move on to classifying them into each class. And both training and testing datasets used, should be in Sinhala Unicode.

The goal of this study was to propose a Named Entity Recognition system that uses Reinforcement Learning techniques to learn. The accuracies and improvements were not the main concern of this study due to the limited time. Hence the scope of this research was to cast the NER task as an RL problem.

1.8 Summary

This chapter laid the foundations for this dissertation. It introduced the NER task and research gap, research questions and hypotheses. Then the research was justified, definitions were presented, the methodology was briefly described and justified, the dissertation was outlined, and the limitations were given. On these foundations, the dissertation can proceed with a detailed description of the research.

Chapter 2 - Literature Review

2.1 Introduction

This chapter mainly describes the current status of the literature in the Named Entity Recognition task and Reinforcement Learning paradigm. This chapter consists of, Background to NER, Supervised Methods, Semi- Supervised methods, Unsupervised Methods,Named Entity Recognition for Sinhala, Background to RL, Summary.

2.2 Background to NER

Named Entity Recognition (NER) is the process of locating a word or a phrase that references a particular entity within a text. The NER task first appeared in the Sixth Message Understanding Conference (MUC6) Sondheim (1995)[12] and involved recognition of entity names (people and organizations), place names, temporal expressions and numerical expressions. In MUC-6, Named entities (NEs) were categorized into three types of label, each of which uses specific attribute for a particular entity type. Entities and their labels were defined as follows:

- ENAMEX: person, organization, location
- TIMEX: date, time
- NUMEX: money, percentage, quantity

Different fine grained or domain dependent annotation schemes have been proposed by many researchers. In practice, it is convenient to work with coarse classification than fine grain classification due to data sparsity.

Several approaches to tackle this task can be seen in the literature. In the section 2.3, supervised methods will be discussed. In the section 2.4, semi supervised methods will be discussed. In the section 2.5, Unsupervised methods will be discussed. In the section 2.6 studies that has been conducted Sinhala Named Entity Recognition will be discussed.

2.3 Supervised Methods

Supervised methods are class of algorithm that learn a model by looking at annotated training examples. Among the supervised learning algorithms for NER, considerable work has been done using Hidden Markov Model (HMM), Decision Trees, Maximum Entropy Models (ME), Support Vector Machines (SVM) and Conditional Random Fields (CRF). Typically, supervised methods either learn disambiguation rules based on discriminative features or try to learn the parameter of assumed distribution that maximizes the likelihood of training data. Each of these methods will be discussed in detail in next sections.

2.3.1 Hidden Markov Models

HMM is the earliest model applied for solving NER problem by Bikel et al. (1999) for English. Bikel introduced a system, IdentiFinder[13], to detect NER. According to Bikel's formulation of the problem in the Identifinder system, only a single label can be assigned to a word in context. Therefore, the model assigns to every word, either one of the desired classes or the label NOT-A-NAME to represent "none of the desired classes". The task is to find the most likely sequence of name-classes(NC) given a sequence of words(W):

$$max\ Pr(NC \mid W)$$

IdentiFinder reported NE accuracy of 94.9% and 90% for a mixed case English (MUC-6 data and a collection of Wall Street Journal documents) and mixed case Spanish (MET-1 data, comprised of articles from news agencies AFP) respectively.

Zhou and Su (2002) [14] modified the IdentiFinder model by using mutual information. Given a token sequence $G_1^n = g_1, g_2, g_3 \ldots g_n$ the goal of the learning algorithm is to find a stochastically optimal tag sequence $T_1^n = t_1, t_2, t_3 \ldots t_n$ that maximizes,

$$Pr(G_1^n \mid T_1^n) = logPr(T_1^n) + \frac{Pr(T_1^n, G_1^n)}{Pr(T_1^n)Pr(G_1^n)}$$

Unlike IdentiFinder, Zhou's model directly generates original NE tags from the output words of the noisy channel. Zhou's model assumes mutual information independence while HMM assumes conditional probability independence. The HMM-based chunk tagger gave an accuracy of 96.6% on MUC-6 data and 94.1% on MUC-7 data.

2.3.2 Maximum Entropy based Model

Maximum entropy model, unlike HMM, are discriminative model. Given a set of features and training data, the model directly learns the weight for discriminative features for classification. In Maximum entropy models, objective is to maximize the entropy of the data, so as to generalize as much as possible for the training data. In ME models each feature is associated with parameter λ_i. Conditional probability is thus obtained as follows:

$$P(f|h) = \frac{\prod_i \lambda_i^{g_i(h,f)}}{Z_\lambda(h)}$$

$$Z_\lambda(h) = \sum_f \prod_i \lambda_i^{g_i(h,f)}$$

Maximizing the entropy ensures that for every feature g_i , the expected value of g_i, according to M.E. model will be equal to empirical expectation of $g_i$in the training corpus. Finally, Viterbi algorithm is used to find the highest probability path through the trellis of conditional probabilities which produces the required valid tag sequences.

The MENE system [15] - Accuracy reported for the MENE system on MUC-7 data is 88.80%.

Curran's ME Tagger [16] - Reported an accuracies of 84.89% for the English test data and 68.48% for the German test data of CoNLL-2003 shared task.

2.3.3 SVM Based Models

Support Vector Machine was first introduced by Cortes and Vapnik (1995) [17] based on the idea of learning a linear hyperplane that separate the positive examples from negative example by large margin. Large margin suggests that the distance between the hyperplane and the point from either instances is maximum. The points closest to hyper plane on either side are known as support vectors.

The linear classifier is based on two parameters, a weight vector W perpendicular to the hyperplane that separates the instances and a bias b which determines the offset of the hyperplane from the origin. A sample x is classified as positive instance if ,

$$f(x) = wx + b > 0$$

and negative otherwise. If the data points are not linearly separable, then a slack is used to accept some error in classification. This prevents the classifier to over fit the data. When there are more than two classes, a group of classifiers are used to classify the instance.

McNamee and Mayfield (2002) tackle the problem as binary decision problem,i.e. if the word belongs to one of the 8 classes, i.e. B- Beginning, I- Inside tag for person, organization, location and misc tags. Thus there are 8 classifiers trained for this purpose. For CoNLL 2002 data, reported accuracies were 60.97 and 59.52 for Spanish and Dutch respectively.

2.3.4 CRF Based Models

Conditional random field were introduced by Lafferty et al. (2001) [18] as a statistical modeling tool for pattern recognition and machine learning using structured prediction. McCallum and Li (2003) [19] proposed a feature induction method for CRF in NE. Let, $o = < o_1, o_2, \ldots\ldots, o_T >$ be some observed input data sequence, such as a sequence of words in a text inside the document (the values on n input nodes of the graphical model). Let S be a set of FSM states, each of which is associated with a label, l L, (such as ORG). Let $s = < s_1, s_2, \ldots\ldots, s_T >$ be some sequence of states, (the values on T output nodes).

By the Hammersley Clifford theorem, CRFs define the conditional probability of a state sequence given an input sequence to be,

$$P(s|o) = \frac{1}{Z} exp(\sum_{t=1}^{T} \lambda_k f_k(s_{t-1}, s_t, o, t))$$

where $Z$is the normalization factor obtained by marginalizing over all state sequences, $f_k(s_{t-1}, s_t, o, t)$is an arbitrary feature function and $\lambda_k$is the learned weight for each feature function. By using dynamic programming, state transition between two CRF states can be efficiently calculated. The modified forward values, $\alpha_T(s_i)$, to be the "unnormalized probability" of arriving state si given the observations $< o_1, o_2, \ldots\ldots, o_T >$. $\alpha_0(s)$is set to probability of starting in each state s, and recursively calculated as :

$$\alpha_{t+1}(s) = \sum_{s'} \alpha_t(s') exp(\sum_k \lambda_k f_k(s', s, o, t))$$

The backward procedure and Baum-Welch have been similarly modified. $Z_0$is given by $\sum_k \alpha_T(s)$. Viterbi algorithm for finding the most likely state sequence given the observation sequence have been 7 modified from its HMM form. Experiments were performed on CoNLL 2003 shared task data, and achieved an accuracy of 84.04% for English and 68.11% for German

2.4 Semi-Supervised methods

Semi supervised learning algorithms use both labeled and unlabeled corpus to create their own hypothesis. Algorithms typically start with small amount of seed data set and create more hypothesis' using large amount of unlabeled corpus. Semi-supervised NER systems will be discussed in the next section.

Carreras et al. (2002) [20] have modeled (NER using AdaBoost) the Named entity identification task as sequence labeling problem through BIO labeling scheme. Input is considered as word sequence to label with one of the Beginning of NE (B-) tag, Inside of tag (I-) and outside of NE (O-) tag. Three binary classifiers are used for tagging, one corresponding to each tag.

Orthographic and semantic features were evaluated over a shifting window allowing a relational representation of examples via many simply binary propositional features.

The binary AdaBoost is used to with confidence rated predictions as learning algorithm for the classifiers. The boosting algorithm combines several fixed-depth decision trees. Each tree is learned sequentially by presenting the decision tree a weighting over the examples which depend on the previous learned trees.

The Spanish data corresponds to the CoNLL 2002 Shared Task Spanish data and shows a performance of 79.28%

2.5 Unsupervised methods

A major problem with supervised setting is requirement of specifying large number of features. For learning a good model, a robust set of features and large annotated corpus is needed. Many languages don't have large annotated corpus available at their disposal. To deal with lack of annotated text across domains and languages, unsupervised techniques for NER have been proposed.

2.5.1 KNOWITALL

KNOWITALL is domain independent system proposed by Etzioni et al. (2005)[21] that extracts information from the web in an unsupervised, open-ended manner. KNOWITALL uses 8 domain independent extraction patterns to generate candidate facts.

For example, the generic pattern "NP1 such as NPLIST2" indicates that the head of each simple noun phrase(NP) in the list of NPLIST2 is a member of class named NP1. It then automatically tests the plausibility of the candidate facts it extracts using point-wise mutual information (PMI) computed using large web text as corpus. Based on PMI score, KNOWITALL associates a probability with every facts it extracts, enabling it to manage the trade-off between precision and recall. It relies on bootstrapping technique that induces seeds from generic extraction patterns and automatically generated discriminator phrases.

2.5.2 Unsupervised NER across Languages

Munro and Manning (2012) have proposed a system that generates seed candidates through local, cross language edit likelihood and then bootstraps to make broad predictions across two languages, optimizing combined contextual, word-shape and alignment models. It is completely unsupervised, with no manually labelled items, no external resources, only using parallel text that does not need to be easily alignable. The results are strong, with F = 0.85 for purely unsupervised named entity recognition across languages, compared to just F = 0.35 on the same 37 data for supervised cross-domain named entity recognition within a language. A combination of unsupervised and supervised methods increases the accuracy to F = 0.88. The tests were done on the parallel corpus of English and Haitian Krreyol text messages used in the 2010 Shared Task for the Workshop on Machine Translation.

2.6 Named Entity Recognition for Sinhala

Considering all these common methods to solve Named Entity Recognition, there are few attempts to apply these methods to indic languages and more specifically Sinhala language.

Although there are successful NER systems for languages such as English, the amount of prior research done on NER for Sinhala is very minimal. The effectiveness of the data driven techniques for Sinhala language is described in the papers Named entity recognition for Sinhala language[22] and Ananya - a Named-Entity-Recognition (NER) system for Sinhala language[23]. J.K.Dahanayaka and A.R.Weerasinghe [22] have tested Sinhala NER performance with both ME and CRF methods and reported the results using a manually annotated corpus. Their work only focused on identifying Named Entity boundaries in a given text, but the classification part (classifying named entities into different classes such as person, location, and organization) was not done. They conclude that the CRF model gives better results compared to ME model.

In a later research, Udayangi and A.R.Weerasinghe [24] discuss a solution based on a hybrid approach for Sinhala NER. The hybrid approach includes an NER system based on CRF and a Rule-based post-processor. The rule-based post-processor makes use of context-based word lists to identify named entities. They have improved the work in [22] by classifying the extracted named entities into three different NE classes (Person names, Locations and Organizations). But the annotated corpus used in their research had no contextual information, because the corpus was created by getting set of words per each class separately (Person names, Locations, Organization and NonNEs), annotate words in each set for the class tag and mixing all of them together. In other words, this corpus contained only named entities.

Structured Prediction or Structured (output) Learning is an umbrella term for supervised machine learning techniques that involves predicting structured objects, rather than scalar discrete or real values [25]. Named Entity Recognition in textual data is a structured prediction problem because the output is a sequence of words which has a structure. All the approaches described above use either supervised or Rule-based or hybrid methods. However, due to lack of resource (Annotated corpus) for Sinhala language, the learning ability of the supervised learning models is limited.

2.7 Background to RL

Markov Decision Problems (MDPs) are sequential decision making problem, in which a system has to make a decision in each decision making state. Richard Bellman showed that MDPs can be effectively solved using Dynamic Programming (DP). However, the classical dynamic programming methods break down on the problems with large-scale and complex MDPs. This is due to the requirement of computing power to solve such problems.

Reinforcement Learning (RL) emerged as a technique of solving these large-scale and complex MDPs in a near-optimal way. The modern science of RL has emerged from combining of notions from four different fields namely, Dynamic Programming, Artificial

Intelligence (AI), stochastic approximation and function approximation (regression, Bellman error, and neural networks).

2.8 Markov Decision Process

MDP is a framework used in stochastic control theory of discrete-event systems. MDPs are driven by underlying Markov chains. In a Markov chain, at each discrete time step, the system jumps from one state to another. The probability of such transitions to the next state depend only on the current state and not on the previous states. In such a setting, at some states, the system has to choose an *action* from a set of predefined actions. The actions to be taken from each state, is called a *policy*. Policy is a mapping from the set of states to the set of actions. The system earns a *reward* in transitioning from one state to another under the influence of the chosen action. Solving an MDP is to find the *optimal* policy that yields the maximum amount of rewards.

2.9 Applications of Reinforcement Learning

RL has been applied in a large number of domains successfully. Here we enumerate a few case studies related to operations management. In particular, we describe the special features of the algorithms that made them suitable for the domain of application. Continuous-time discounted algorithms[26] were employed for elevator scheduling[27] because the problem structure had a continuous-time Markov chain underlying it. The job-shop scheduling problem in Zhang and Dietterich (1995) had an episodic nature, and hence TD(λ) became preferable[28].

The AGV routing problem in Tadepalli and Ok (1998)[29] is one of the few case studies of model-building RL for large-scale problems. The model they learn is able to capture the complex dynamics of the AGV problem. A well-known "revenue management problem"[30] can be set up as an average-reward SMDP. The work related to hyper

heuristics can be used when RL is to be used dynamically to select a meta-heuristic. An SMDP with a discount factor was employed for a cell phone network-management problem [31] that allowed handy combination with a neuron-based function approximator. The retailer-inventory management problem[32] used regular Q-Learning with a vanishing discount factor. It is likely that the field of applied RL will explode in the coming years because of RL's ability to solve problems previously considered intractable.

2.10 Summary

Throughout the literature many solutions for NER can be found. Most of the solutions work with a significant amount of accuracy for languages such as English which is well resourced. However, for Sinhala, due to its inherent features such as lack of capitalization, such a system cannot be used.

There are few research conducted for Sinhala NER. Most of them use rule based methods, or supervised learning methods. No attempt can be found where RL has been applied to solve NER for Sinhala.

Chapter 3 - Design

3.1 Introduction

This chapter mainly elaborates the proposed solutions to the research problem. It consists of four sections, namely; Research Design, NER as a Structured Prediction problem, Reduction to Reinforcement Learning and the Reinforcement Learner.

3.2 NER as a Structured Prediction problem

Most of the existing solutions for Sinhala Named Entity Recognition use the information about the word that is in consideration, or a word window that includes the said word. However, the information about the structure can be exploited in order to have more insights, in a sequence tagging problem. Capturing the structure or the dependency between the output variables is considered important as much as the relationship between the input and the output structured prediction. Figure 3.1 shows the initial problem perception.

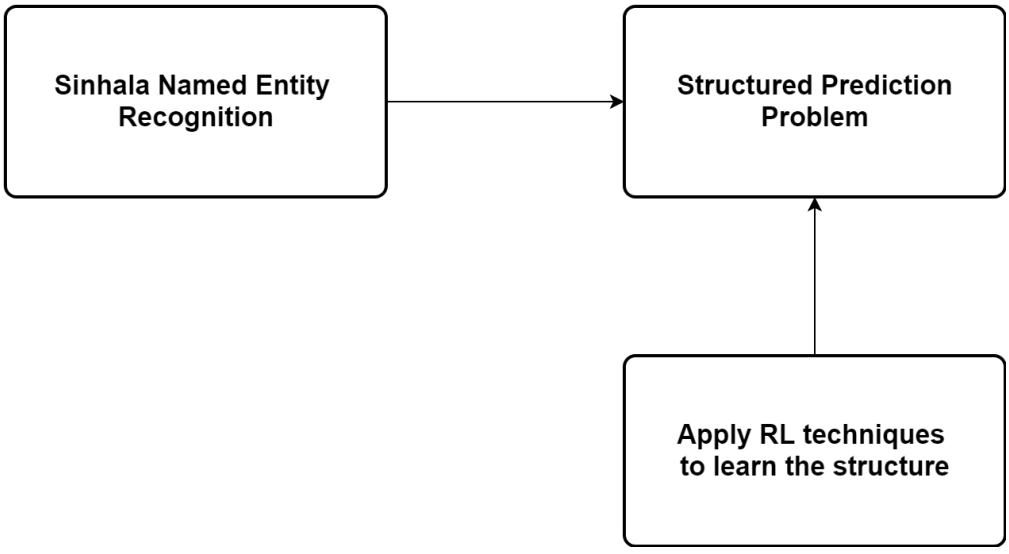

Figure 3.1 - Initial Problem Perception

3.3 Reduction to Reinforcement Learning

In order to apply RL techniques, the problem should be converted to an RL problem. In an RL problem, there are States that an RL agent can explore, Actions that an RL agent can perform, and Rewards that an agent receive based on the state and the action it is in. Following the work, Structured Prediction with reinforcement learning [16] of Francis et al., the Named Entity Recognition task has been reformed as a Markov Decision Problem. After the conversion, a learning algorithm has been proposed which lets the RL agent learn from exploring and rewards. Figure 3.2 shows the high-level approach of the study.

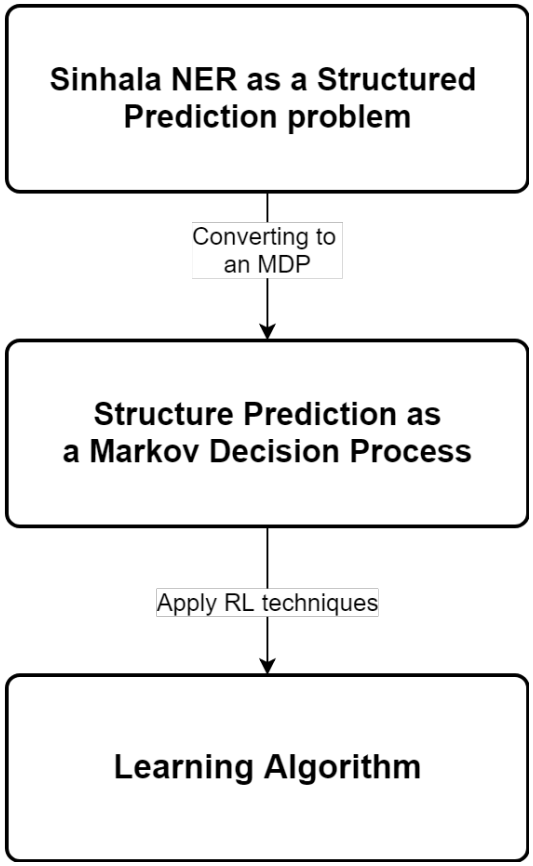

Figure 3.2 - High-level approach of the study

The main components of an RL problem has been defined as following,

• **States**

For this study, it is assumed that the input sentences are of fixed length. For each sentence, initially a state space is generated. Figure 3.3 shows such an induced search space for a binary classification.

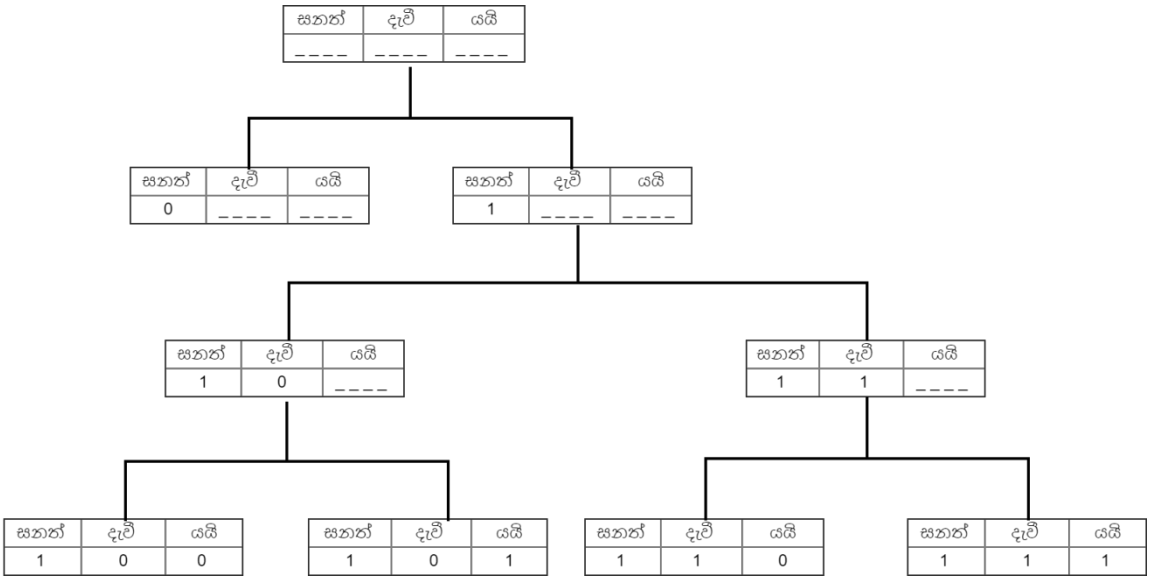

Figure 3.3 - Induced State Space for a Binary Classification

Each state contains both an input x and a partial output $\underline{y}$. Let $\underline{Y}$ be the set of all possible partial outputs. The set of states is then S = X×$\underline{Y}$. There is one initial state per possible input x: s initial(x) = (x, $\underline{y_\epsilon}$) where $\underline{y_\epsilon}$ ϵ $\underline{Y}$ is the initial empty solution. Each state except for leaf nodes, has two children, one for each action in binary classification. For a multiclass problem, the states in the induced tree may contain a number of children equal to the number of classes. Final states contain the complete outputs where each word in the input sequence has a tag.

• **Actions**

Actions are defined as elementary modifications of the partial output $\underline{y}$. For example, Partial outputs are partially labeled sequences and an elementary modification might be the addition of a single label prediction to the current partial output. An action can be denoted as $A_s \subset A$ where A is the set of actions available in state s. In this study, only the binary classification has been considered, thus only actions in a particular state are adding the next label as a Named Entity or Non-Named Entity. However this can be extended by adding more actions. i.e.: Next label is a Person or a Location or an Organization or a Non-Named Entity.

• **Transitions**

Transitions are deterministic and replace the current partial output by the transformed partial output. Transitions do not change the current input: $T((x, \underline{y}), a) = (x, a(\underline{y}))$ where $a(\underline{y})$ denotes the partial output modified by action a.

• **Rewards**

Rewards are given for each decision step. Initially, when the reference policy is generated, all the costs for actions are generated. Figure 3.4 shows the induced tree with the calculated costs for each action for a particular input.

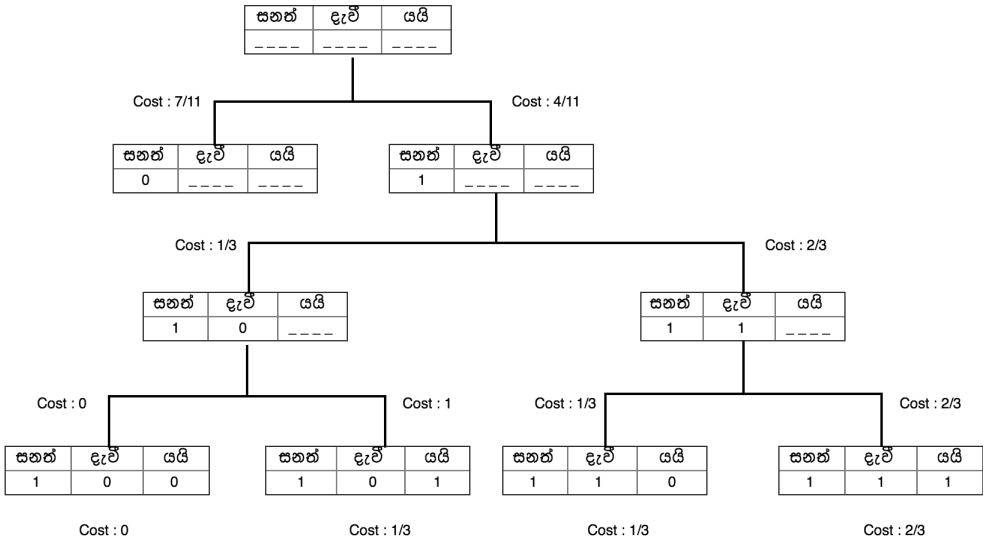

Figure 3.4 – Induced tree with cost values

In the figure, Non-Named Entity (NNE) labels are denoted with 0 and Named Entities(NE) are denoted with 1. From each state, the left branch shows tagging the next word as an NNE and the right branch shows tagging the next word as an NE. The cost for each action for a 3-word sentence is depicted above with the cost values. The cost for leaf states are calculated as follows.

$$cost(s) = \frac{\left|\{\underline{y}_i \mid \underline{y}_i = y_i\}\right|}{|y_i|}$$

where s is a leaf state, $\underline{y}_i$ is the i^{th} label in the partial output $\underline{y}$, and y_i is the i^{th} label in the true label set y. Cost function for non-leaf nodes are defined as follows.

$$cost(s,a) = \frac{\sum_{i=1}^{n} cost(T(s,a),a_i)}{\sum_{i=1}^{n} \sum_{j=1}^{n} cost(T(s,a_i),a_j)}$$

where s is not a leaf state, T is the transition function and $T(s,a)$ denotes the next state when the action a is taken from state s and $A = \{a_1, a_2, a_3, \ldots . a_n\}$ is the set of actions. For a binary classifications, there two actions for each state.

3.4 Neural Net

In a simple reinforcement learning problem, a table can be used to store the learned information about the particular problem. However in an NER problem, using a table to look up such information is impractical since the number of states can be large and it has to deal with unseen inputs. Hence, an approximator should be used to learn the structure. For this particular task, a simple multilayer perceptron has been used as the function approximator. Figure 3.5 shows the network architecture of the neural network.

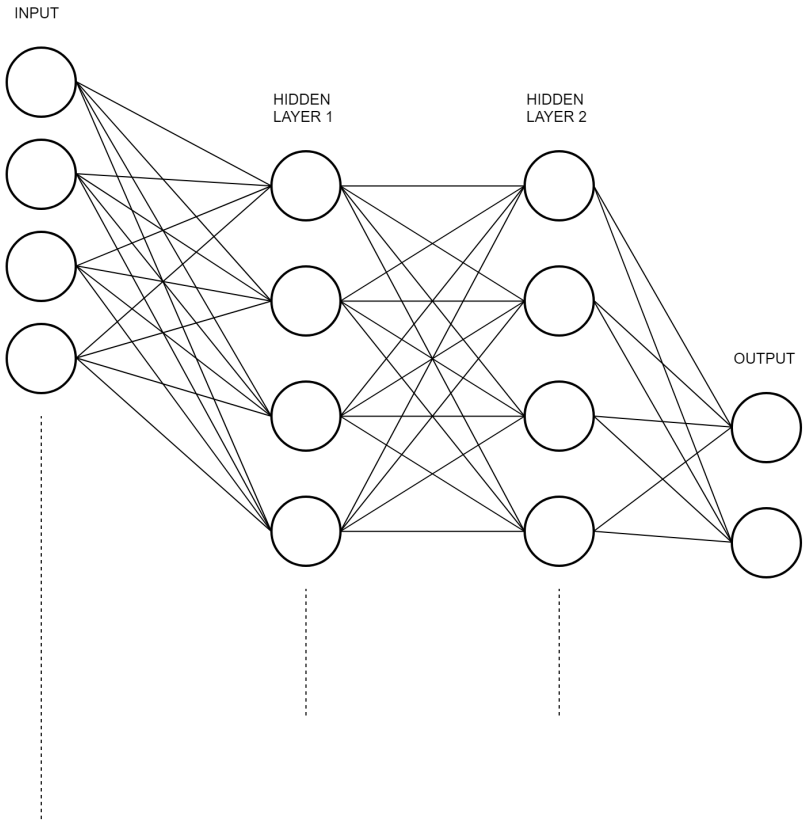

Figure 3.5 - Network architecture of the neural network

3.5 Reinforcement Learner

The final system consists of a Learning algorithm and a Neural Network. Learner takes sentences as inputs, and for each input, it generates an induced state space, a reference policy which contains costs for each actions at each state using correct labels and a learned policy using the Neural network. Learner then explores the state using the learned policy, and collects cost examples as each state using a combination of learned and reference policy. The cost examples is then returned to the experience set, and then randomly selected. Figure 3.6 shows the diagram of the research design.

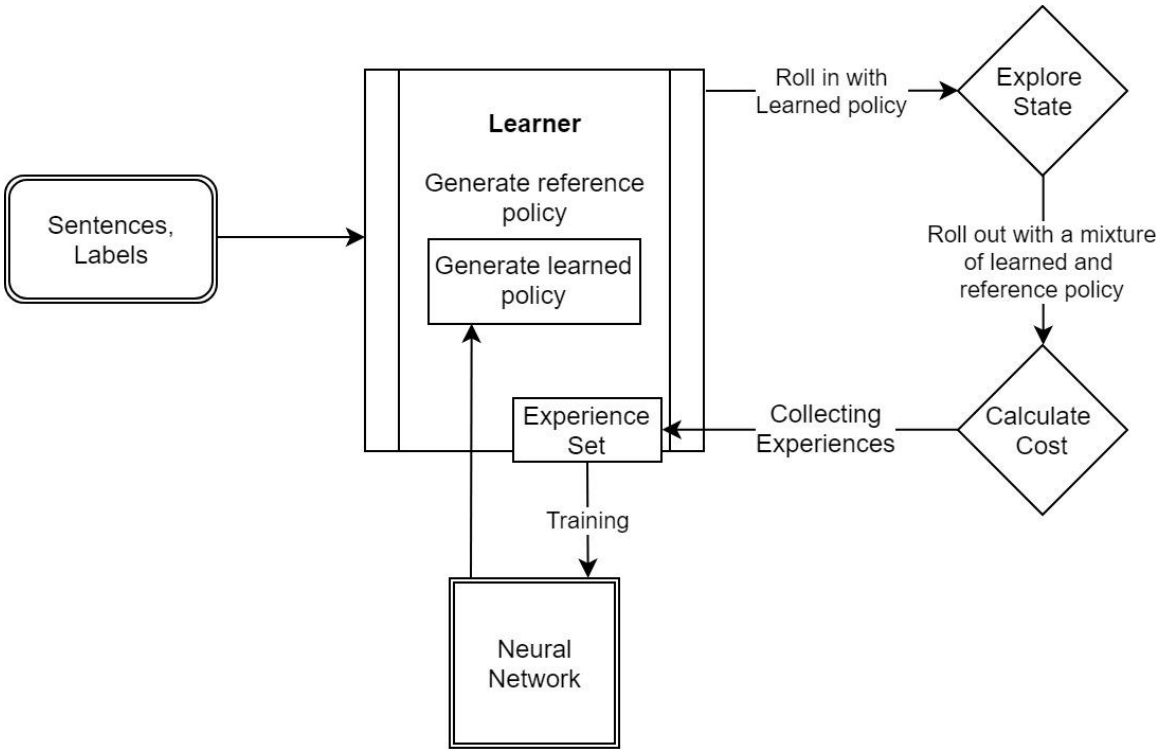

Figure 3.6 - Research Design

Chapter 4 - Implementation

4.1 Introduction

This chapter mainly elaborates the implementation details of this study. It consists of seven sections, namely; Learning Algorithm, Learner, Induced State Space, Reference Policy, Learned Policy, Feature Function, Embedding Builder, Neural Network and Summary.

4.2 Learning Algorithm

Figure 4.1 shows the learning algorithm of the Reinforcement Learner.

Initialize a policy π_0.
for all $i \in \{1, 2, \ldots, N\}$ (loop over each instance) **do**
  Generate a reference policy π^{ref} based on $\boldsymbol{y}_i$.
  Initialize $\Gamma = \emptyset$.
  for all $t \in \{0, 1, 2, \ldots, T-1\}$ **do**
    Roll-in by executing $\pi_i^{\text{in}} = \hat{\pi}_i$ for t rounds and reach s_t.
    for all $a \in A(s_t)$ **do**
      Let $\pi_i^{\text{out}} = \pi^{\text{ref}}$ with probability β, otherwise $\hat{\pi}_i$.
      Evaluate cost $c_{i,t}(a)$ by rolling-out with π_i^{out} for $T - t - 1$ steps.
    end for
    Generate a feature vector $\Phi(\boldsymbol{x}_i, s_t)$.
    Set $\Gamma = \Gamma \cup \{\langle c_{i,t}, \Phi(\boldsymbol{x}_i, s_t)\rangle\}$.
  end for
  $\hat{\pi}_{i+1} \leftarrow \text{Train}(\hat{\pi}_i, \Gamma)$ (Update).
end for
Return the average policy across $\hat{\pi}_0, \hat{\pi}_1, \ldots \hat{\pi}_N$.

Figure 4.1 - Learning algorithm of the Reinforcement Learner

Input for the algorithm, is a set of sentences x_i of length 10 and their corresponding tag sequence y_i , $\{\{x_1, y_2\}, \{x_2, y_2\}, \ldots\ldots, \{x_{n-1}, y_{n-1}\}, \{x_n, y_n\}\}$. The algorithm loops over the set, and for each input x_i , it generates an action sequence (learned policy) to reach a leaf state using the cost estimations given the neural network. Then it generates the actual cost values for each action(reference policy) using the true label set y_i . Initially, the network will output random cost values after the random initialization. For each action in learned policy, the algorithm will roll-out using a combination of the learned policy and the reference policy to reach a leaf state and will collect a cost example for the particular state. After the end of the loop over actions of learned policy, it will send the collected set of cost examples to train the Neural Network.

4.3 Learner

The learning algorithm is implemented in the class Learner. It consists of the learning algorithm, Neural Network, and other required components of the system. Following code snippet shows the implementation of the learning algorithm.

```
from induced_tree import InducedTree
import tensorflow as tf
from gensim.models import Word2Vec
import numpy as np
import random
class Learner:
   def learn(self, beta):
       import numpy as np
       training_data = self.training_data
       training_labels = self.training_labels
       actions = self._actions
       for i in range(len(training_data)):
           self._induced_tree = self._induce_tree(training_data[i])
           learned_policy = self._generate_learned_policy()
           reference_policy = self._generate_reference_policy(self._induced_tree,
training_labels[i])
           experience = []
           s = 0
           for j in range(len(training_data[i])):
```

```
                costs = []
                min_cost = 1
                for action in actions:
                    choice = int([np.random.choice([0, 1],
                                                   1,
                                                   p=[beta, 1 - beta])][0][0])
                    if choice == 0:
                        policy = reference_policy
                    else:
                        policy = learned_policy
                    action_cost = self._roll_out(policy, choice, s, actions[action],
self._induced_tree, training_labels[i])
                    costs.append(action_cost)
                feature_vector = self._generate_feature_vector(s)
                experience.append([feature_vector, costs])
                s = learned_policy[s]

            sample = self._sample(experiences)
            self._train_classifier(sample, i)
```

4.3 Induced State Space

The state space is generated using a tree data structure. Two classes namely, Node and InducedTree, have been used to build the state space. Nodes store the necessary information and InducedTree class creates the data structure to hold the Nodes. Following code snippet shows the Node class.

```
class Node:
    def __init__(self, sentence, labels):
        self._sentence = sentence
        self._labels = labels
        self._cost = 0
        self._optimal_action = None
        self._action_cost = [None, None]


    def get_sentence(self):
        return self._sentence

    def get_labels(self):
        return self._labels

    def set_labels(self, labels):
```

```
        self._labels = labels

    def set_optimal_action(self, action):
        self._optimal_action = action

    def get_optimal_action(self):
        return self._optimal_action

    def set_cost(self, cost):
        self._cost = cost

    def get_cost(self):
        return self._cost

    def get_action_cost(self, action):
        if action == 0 or action == 1:
            return self._action_cost[action]

    def set_action_cost(self, action, cost):
        if action == 0 or action == 1:
            self._action_cost[action] = cost
```

Following code snippet shows the InducedTree class.

```
class InducedTree:
    def __init__(self, sentence):
        """
        dictionary = { 1: node, 2: node ..... }
        node = {_sentence : sentence,
                _labels : partial_labels,
                _cost: [ 0: left_cost, 1: right_cost],
                _optimal_action: 0 or 1
                }
        relations = { 1: [left child of 1, right child of 1], 2: [left child of 2,
right child of 2]}
        """
        self._dictionary = {}
        self._relations = {}
        self._induce_tree(self._dictionary, self._relations, sentence)

    def _induce_tree(self, dictionary, relations, sentence):
        length = len(sentence)
        for i in range(2 ** (length+1)):
            node = Node(sentence, [])
            dictionary[i] = node
        count = 1
        for i in range(2 ** length - 1):
```

```
            relations[i] = [count]
            count += 1
            relations[i].append(count)
            count += 1
        self._generate_labels(dictionary, length)

    def _generate_labels(self, dictionary, levels):
        state = 0
        dictionary[0].set_labels([None] * levels)
        for level in range(1, levels + 1):
            num_of_nodes = 2 ** level
            num_of_nones = levels - level
            for node in range(num_of_nodes):
                state += 1
                labels = bin(node)[2:].zfill(level)
                labels = list(map(int, list(labels))) + num_of_nones * [None]
                dictionary[state].set_labels(labels)

    def get_right_child(self, state):
        if state in self._relations:
            return self._relations[state][1]
        return None

    def get_left_child(self, state):
        if state in self._relations:
            return self._relations[state][0]
        return None

    def get_node(self, state):
        if state in self._dictionary:
            return self._dictionary[state]
        else:
            return None

    def is_leaf(self, state):
        if self.get_left_child(state) is None and self.get_right_child(state) is None:
            return True
        else:
            return False

    def get_dictionary(self):
        return self._dictionary

    def transition(self, state, action):
        if self.is_leaf(state):
            return None
        elif action == 0:
```

```
            return self.get_left_child(state)
        elif action == 1:
            return self.get_right_child(state)
        else:
            raise ValueError("Invalid value for action")
```

4.4 Reference policy

The reference policy is generated recursively. First the leaf node costs are calculated and stored. The action corresponds to the least cost value is marked as the reference policy action corresponds to that state. Then the cost values for ancestors are generated recursively in a bottom up manner. All the states are marked with an action that leads to the subtree with the least cost. Then the optimal action sequence from any state to a leaf state, is returned as the reference policy. Following code snippet shows the generation of the reference policy.

```
    def _generate_reference_policy(self, induced_tree, labels):
        self._generate_node_costs(induced_tree, labels)
        policy = self._get_reference_policy(induced_tree)
        return policy

    def _generate_node_costs(self, tree, labels):
        current = 0
        self._recurse(current, tree, labels)

    def _recurse(self, state, tree, labels):
        if tree.is_leaf(state):
            node = tree.get_node(state)
            partial_labels = node.get_labels()
            correct_count = 0
            for j in range(len(partial_labels)):
                if partial_labels[j] == labels[j]:
                    correct_count += 1
            cost = 1 - (float(correct_count) / len(labels))
            node.set_cost(cost)
            return len(labels) - correct_count
        else:
            node = tree.get_node(state)
```

```
            left = self._recurse(tree.get_left_child(state),
                                 tree, labels)
            right = self._recurse(tree.get_right_child(state),
                                  tree, labels)
            total = float(left + right)
            node.set_action_cost(0, left / total)
            node.set_action_cost(1, right / total)
            return total

    def _get_reference_policy(self, tree):
        tree_dict = tree.get_dictionary()
        policy = {}
        for i in tree_dict:
            action = tree_dict[i].get_optimal_action()
            if action == 0:
                s = tree.get_left_child(i)
            else:
                s = tree.get_right_child(i)
            policy[i] = s
        return policy
```

4.5 Learned policy

The learned policy is generated using the Neural Network output. The states are transformed using a feature function and fed into the Neural Network. The Neural Network outputs the costs for each action. The algorithm will choose the action with the minimum cost for each state and returns the optimal action sequence from any state to a leaf state. Following code snippet shows the Learned policy generation.

```
def _generate_learned_policy(self):
    learned_policy = {}
    tree = self._induced_tree
    current_state = 0
    while not tree.is_leaf(current_state):
        current_feature_vector = self._generate_feature_vector(current_state)
        action = self.predict(current_feature_vector)
        action = np.array(action).tolist()[0]
        if action is 0:
          learned_policy[current_state] = tree.get_left_child(current_state)
```

```
            current_state = tree.get_left_child(current_state)
        else:
            learned_policy[current_state] = tree.get_right_child(current_state)
                  current_state = tree.get_right_child(current_state)
        return learned_policy
```

4.6 Feature function

For this particular study, the state has been encoded using a trigram word window. This function can be customized. The only constraint is, the output dimension of the feature function should match the input dimension of the Neural Network. In this study, the feature function will take the word embeddings of the three words namely, previously tagged word, current word, and the next word, and combine them together to encode the state. The reason behind this choice was, word embedding captures relationships between the words using the context. As the study was more focused on the structure of the sentence, the information about the relationships between words were important.

The word embedding dimension is decided as 7 as the vocabulary size of the training corpus is approximately 2500. This was decided following a general rule of thumb, which is the 4th root of the size of the vocabulary should be the size of the embedding dimension. Following code snippet shows the feature function. To generate the word embeddings, Fast text algorithm of the gensim python library has been used. The word embedding model has to be built prior to the training using the corpus. Embedding builder class is explained in the section 4.7.

```
def _generate_feature_vector(self, state):
    word2vec_model = Word2Vec.load(self._model)
    partial_labels = self._induced_tree.get_node(state).get_labels()
    sentence = self._induced_tree.get_node(state).get_sentence()
    word_index = -1
    for i in range(len(partial_labels)):
       if partial_labels[i] is None:
           word_index = i
           break
```

```
        previous_word_vector = [0. for i in range(50)]
        next_word_vector = [0. for i in range(50)]
        if word_index > 1:
          previous_word = sentence[word_index-1]
          previous_word_vector = self._get_word_embedding(word2vec_model.wv,
                                                          previous_word)
        previous_word_vector = np.array(previous_word_vector).tolist()
        current_word = sentence[word_index]
        if word_index < len(sentence) - 1:
          next_word = sentence[word_index+1]
          next_word_vector = self._get_word_embedding(word2vec_model.wv,
                                                      next_word)
        next_word_vector = np.array(next_word_vector).tolist()
        current_word_vector = self._get_word_embedding(word2vec_model.wv,
                                                       current_word)
        current_word_vector = np.array(current_word_vector).tolist()
        vector = previous_word_vector + current_word_vector +
                 next_word_vector
        return vector


    def _get_word_embedding(self, embedding_model, word):
        try:
            return embedding_model[word]
        except:
            return [0. for i in range(50)]
```

4.7 Embedding Builder

Python gensim library has been used to build the embedding model used in the feature function. As mentioned in the section 4.6, the dimension of the embedding has been decided as 7, and the size of the context window has been decided as 3. The sentences of length 10 has been obtained from the input file ignoring the '.'.

```
import codecs
from gensim.models import FastText
class Word2VecBuilder:
    def __init__(self, inputfile, outputfile,
                 size=7, window=3, min_count=1, workers=4, sg=1):
        with codecs.open(inputfile, 'rb',
                         encoding='utf-16', errors='ignore') as infile:
            fin = infile.read()
        lines = [[]]
```

```
        word = ""
        for letter in fin:
            if letter == "\t":
                if not word == "":
                    lines[len(lines) - 1].append(word)
                    word = ""
                continue
            elif letter == ".":
                lines.append([])
                continue
            elif not (u'\u0d80' <= letter <= u'\u0dff'):
                continue
            word += letter
        model_ted = FastText(lines, size=size, window=window,
                             min_count=min_count, workers=workers, sg=sg)
        model_ted.save(outputfile)
```

4.8 Neural Network

Since this study focuses on the conversion of the NER task to an RL problem and lay out the groundwork that future studies can improve upon, a simple Neural Network has been used to learn the cost function. As mentioned in the design chapter, the Neural Network has an input layer, on hidden layer and an output layer. The input later dimension is decided as 21 in order to match the dimension of 3 word embeddings. The output dimension is decided as 2 in order to match the number of classes. And the dimension of the hidden layer is decided as 12, following a general rule of thumb, the number of neurons in the hidden layer should be the mean of the number of neurons in the input layer and the output layer. The Neural Network has been implemented using Tensorflow python library.

4.8.1 Initialization

Following code snippets show the Neural Network initialization. It creates a multilayer perceptron with one hidden layer and uses AdamOptimizer to optimize the backpropagation algorithm.

```
def _init_classifier(self):
    self._x = tf.placeholder("float", [None, self._n_input])
    self._y = tf.placeholder("float", [None, self._n_classes])
    weights = {
       'h1': tf.Variable(tf.random_normal([self._n_input,
                                self._n_hidden_1])),
       'out': tf.Variable(tf.random_normal([self._n_hidden_1,
                                self._n_classes]))
    }

    biases = {
       'b1': tf.Variable(tf.random_normal([self._n_hidden_1])),
       'out': tf.Variable(tf.random_normal([self._n_classes]))
    }

    self._weights_biases['weights'] = weights
    self._weights_biases['biases'] = biases
    self._pred = self._multilayer_perceptron(tf, weights, biases)
    self._cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=self._pred,
                                                       labels=self._y))
    self._optimizer =
tf.train.AdamOptimizer(learning_rate=self._learning_rate).minimize(self._cost)
    pred = tf.nn.softmax(self._pred)
    self._prediction = tf.argmin(pred, 1)
    init = tf.global_variables_initializer()
    self._tf_session = tf.Session(config=tf.ConfigProto())
    self._tf_session.run(init)
```

4.8.2 Training

Following code snippet shows the implementation of the training. This method is called in the learn for every sentence of the input after collecting cost examples for each sentence.

```
    def _train_classifier(self, experience, counter):
        inp = experience[0]
        inp = np.array([np.array(xi) for xi in inp])
        labels = experience[1]
        labels = np.array([np.array(xi) for xi in labels])
        _, c = self._tf_session.run([self._optimizer, self._cost],
                                    feed_dict={self._x: inp,
                                    self._y: labels})
```

4.8.3 Testing

Following code snippet shows the implementation of the testing. This method is called from the main file after the training is finished.

```
    def test_classifier(self):
        X_test = self.testing_data
        print("test size", len(X_test))
        Y_test = self.testing_labels
        accuracy = 0.
         for i in range(len(X_test)):
            prediction = self.predict_labels(X_test[i])
            correct_count = 0.
            for j in range(len(prediction)):
                if prediction[j] == Y_test[i][j]:
                    correct_count += 1
            accuracy += (correct_count/len(prediction))
        accuracy = accuracy/len(X_test)
        return accuracy
```

4.9 Summary

This chapter elaborated the implementation details of the proposed solution and justified the implementation decision. The goal of this study was to cast the NER task as a Reinforcement Learning problem. Thus the implementation has been more focused on the integrating all the required components and build a simple, understandable, and customizable code that can be used in the same task, or in other structured prediction tasks.

Chapter 5 - Results and Evaluation

5.1 Introduction

This chapter mainly elaborates the evaluation details and the results of this study. It consists of four sections, namely; Dataset, High Level Evaluation Design, Evaluation Strategy and Results, Discussion of the result and Discussion of the model.

5.2 Dataset

A Part Of Speech (POS)-tagged corpus prepared by LTRL (Language Technology Research Laboratory) of UCSC (University of Colombo School of Computing) has been used for training and testing the Reinforcement Learner. It consists of about 75,000 Sinhala words, collected from publisher content and archived web content. Each word and punctuation marks having the appropriate Part Of Speech tag label. This POS tagged corpus is retagged for maximal NE's and used as training dataset.

5.3 High Level Evaluation Design

Evaluation of this study has been carried out by splitting the data set mentioned in the section 5.2 into training and testing sets. It is strictly ensured not to use training data for testing. The training data set consist of about 68205 NE annotated Sinhala words while the test set has 5902 Sinhala words extracted from the corpus randomly. The NER system has been trained with the prepared training data set. The evaluation is done by comparing the system output with the output of human linguists. The test set was used to complete that task. Most evaluation systems require exact match on both boundary and entity type. The share task for CONLL 2003 (Sang and Meulder, 2003) is one of the examples for the exact matching. However, in this study, only the individual words and binary tagging has

been the main focus. For instance, “සනත් ජයසූරිය” was treated as two separate named entities. Thus, the boundaries were ignored.

5.4 Evaluation strategy and results

The study has been focused on converting the Named Entity Recognition task to a Reinforcement Learning problem and developing a customizable code base that can be adjusted and experimented on. Thus the most suitable strategy was to evaluate the learning capability of the system with the change of the size of the data set it has been trained on.

The training corpus has 7452 sentences of length 10. The training corpus has been split into 10 training sets. The sets included 100, 250, 500, 1000, 2000, 3000, 5000, 6000, 7452 sentences respectively. For each training set, the testing was done using the previously mentioned test set of approximately 6000 words.

In order to measure the stability of the learning algorithm, for each training set, accuracies of 10 iterations was taken. Figure 5.1 shows the scatter plot of the accuracies against the iteration number.

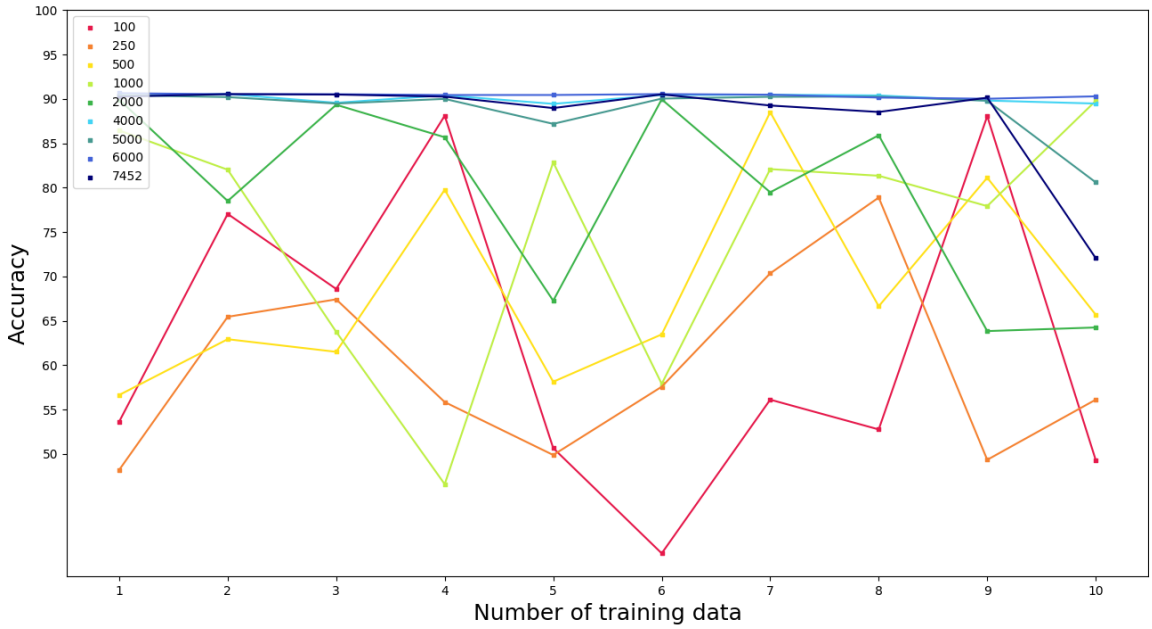

Figure 5.1 Accuracies for each training set for 10 iterations

Each line represents a training dataset. And for each line, a point represents the accuracy at each iteration.

5.5 Discussion of the results

As shown in the figure 5.1, the accuracy has been increased with respect to the size of the data set. The accuracy for the smaller training sets were near random and the variance of the accuracies were also high. It is clear that, as the training data set gets bigger, accuracies improve and stabilize.

However, since the data set contains about 66955 Non-Named Entities and only about 7562 Named Entities, the accuracy yield of the learning algorithm tends to saturate around 90%. This is because, it starts to learn that the likelihood of a word being a Non Named Entity is significantly higher than it being a Named Entity. Since this study only used Word Embedding encoding as the input, the learner has not been able to exploit the language dependent features such as “මයා”, “හිමි”, ”මහත්මිය”.

5.6 Discussion of the model

The present experiments were able to achieve appreciable accuracies. However, the main goal of, mapping between the NLP task of Named Entity Recognition and the Reinforcement Learning paradigm, has yielded a model that is able to be reused as more data becomes available. The source code has been developed in a way such that the main components of the system namely, the feature function, the reward function and the neural network, can easily be updated or eve completely replaced.

Chapter 6 - Conclusion

6.1 Introduction

This chapter mainly elaborates the conclusion of this research. It consists of four sections, namely; Conclusion about the research question, Conclusion about the research problem, Limitations and Implications for further research.

6.2 Conclusion about the research questions

The first research question of this study was, how to cast Sinhala Named Entity Recognition task as a Reinforcement Learning problem. In the literature there has been many studies conducted in the area of NER and RL. Although, the amount of studies conducted on studies which attempt to bridge the gap between these two areas. This study has addressed this gap. And the study has shown that such a mapping is possible and the approximations of Reinforcement Learning techniques can be applied to Named Entity Recognition.

However the second research question has not been answered to the extent where it would achieve state of the art results for Sinhala Named Entity Recognition. However, this study has laid the groundwork for such improvement in future researches by providing a learning algorithm and a simple and customizable implementation of the learning algorithm.

6.3 Conclusion about the research problem

As a low resourced language, it is difficult to find good solutions using data driven techniques for Natural Language Processing tasks for Sinhala. Named Entity Recognition is an important aspect of Natural Language Processing, and having low resources, it is difficult to create a very accurate Machine Learning system for this task. Given the recent resurgence of Reinforcement Learning, it was assumed that Reinforcement Learning

techniques are able to learn to generalize better than other learning techniques. However, to compare the learning capabilities of these techniques a Reinforcement Learning system for Sinhala Named Entity Recognition was necessary.

This research problem has been addressed by this study and it has proposed a complete Reinforcement Learning System. The implementation of this study can be customized and used as a test bed for future researches. The source code of the implementation will be made publicly available.

6.4 Limitations

This study has given the importance on proposing a mapping between Sinhala Named Entity Recognition task and Reinforcement Learning. Thus, for the system, a simple feature function, a simple neural network, and a simple reward function has been used.

Though it can be conveniently extended to classifying Named Entities into Person, Location, and Organization, only the binary classification for NE's and Non NE's has been considered.

6.5 Implications for further research

The work that has been done in this study could be further improved by introducing new feature functions that best suit for Sinhala. And the Neural Network can be replaced to fit the amount of training data that is available. These adjustments can be done in a trial and error manner and improve upon the implementation that will be made publicly available.

References

[1] K. Chang, A. Krishnamurthy, A. Agarwal, H. Daumé III and J. Langford, "Learning to Search Better Than Your Teacher", arXiv.org, 2019. [Online]. Available: http://arxiv.org/abs/1502.02206. [Accessed: 07- Jan- 2019].

[2] Nancy Chinchor, Erica Brown, Lisa Ferro, and Patty Robinson, "1999 Named Entity Recognition Task Definition," MITRE and SAIC, 1999.

[3] Lev Ratinov and Dan Roth, "Design Challenges and Misconceptions in Named Entity Recognition," in CoNLL '09 Proceedings of the 13th Conference on Computational Natural Language Learning, Stroudsburg, PA, 2009, pp. 147-155.

[4] GuoDong Zhou and Jian Su, "Named Entity Recognition Using an HMM-Based Chunk Tagger," in ACL '02 Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, Stroudsburg, PA, 2002, pp. 473-480.

[5] Jenny Rose Finkel and Christopher D. Manning, "Joint Parsing and Named Entity Recognition," in NAACL '09 Proceedings of Human Language Technologies, Stroudsburg, Pa, 2009, pp. 326-334.

[6] Hirotaka Funayama, Tomohide Shibata, and Sadao Kurohashi, "Bottom-Up Named Entity Recognition Using a Two-Stage Machine Learning Method," in MWE '09 Proceedings of the Workship on Multiword Expressions: Identification, Interpretation, Disambiguation, and Applications, Stroudsburg, PA, 2009, pp. 55-62.

[7] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang, "Named Entity Recognition Through Classifier Combination," in CONLL '03 Proceedings of the 7th Conference on Natural Language Learning, vol. 4, Stroudsburg, PA, 2003, pp. 168-171.

[8] Hai Leong Chieu and Hwee Tou Ng, "Named Entity Recognition: A Maximum Entropy Approach Using Global Information," in COLING '02 Proceedings of the 19th International Conference on Computational Linguistics, vol. 1, Stroudsburg, PA, 2002, pp. 1-7.

[9] Andrew McCallum and Wei Li, "Early Results for Named Entity Recognition with Conditional Random Fields, Feature Induction, and Web-Enhanced Lexicons," in CONLL '03 Proceedings of the 7th Conference on Natural Language Learning, vol. 4, Stroudsburg, PA, 2003, pp. 188-191.

[10] Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Frederick Reiss, and Shivakumar Vaithyanathan, "Domain Adaption of Rule-Based Annotators for Named Entity Recognition Tasks," in EMNLP '10 Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, Stroudsburg, PA, 2010, pp. 1002-1012.

[11] Vijay Krishnan and Christopher D. Manning, "An Effective Two-Stage Model for Exploiting Non-Local Dependencies in Named Entity Recognition," in ACL-44 Proceedings of the 21st International Conference on Computational Linguistics, Stroudsburg, PA, 2006, pp. 1121- 1128.

[12] Sixth Message Understanding Conference (MUC-6): Proceedings of a Conference Held in Columbia, Maryland, November 6-8, 1995

[13] D. Bikel, R. Schwartz and R. Weischedel, Machine Learning, vol. 34, no. 13, pp. 211-231, 1999.

[14]G. Zhou and J. Su, "Named entity recognition using an HMM-based chunk tagger", Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02, 2001.

[15]Borthwick, A., Sterling, J., Agichtein, E., & Grishman, R. (1998). Description of the MENE named entity system as used in MUC-7. In Proceedings of the Seventh Message Understanding Conference (MUC-7), Fairfax, Virginia, April 29-May 1, 1998

[16] Curran and S. Clark, "Language independent NER using a maximum entropy tagger", Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 -, 2003.

[17] C. Cortes and V. Vapnik, "Support-vector networks", Machine Learning, vol. 20, no. 3, pp. 273-297, 1995.

[18] John Lafferty, Andrew McCallum, and Fernando C.N. Pereira, "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data", . June 2001.

[19] A. McCallum and W. Li, "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons", Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 -, 2003.

[20] X. Carreras, L. Màrquez and L. Padró, "A simple named entity extractor using AdaBoost", Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 -, 2003.

[21] M. Cafarella, D. Downey, S. Soderland and O. Etzioni, "KnowItNow", Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing - HLT '05, 2005.

[22] J.K. Dahanayaka, A.R. Weerasinghe, "Named Entity Recognition for Sinhala Language", in 2014 International Conference on Advances in ICT for Emerging Regions (ICTer), UCSC Building Complex, No. 35, Reid Avenue, Colombo 7, Sri Lanka., 2014.

[23] S. Manamini, A. Ahamed, R. Rajapakshe, G. Reemal, S. Jayasena, G. Dias, S. Ranathunga, "Ananya-a Named-Entity-Recognition (NER) system for Sinhala language", Moratuwa Engineering Research Conference (MERCon), pp. 30-35, 2016, 2016.

[24] K. Udayangi and A. Weerasinghe, "A Hybrid Approach for Named Entity Recognition."

[88] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T. and Hassabis, D. (2017). Mastering the game of Go without human knowledge. Nature, 550(7676), pp.354-359.

[25] "Structured prediction", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Structured_prediction. [Accessed: 07- Jan- 2019].

[26] A. Barto, S. Bradtke and S. Singh, "Learning to act using real-time dynamic programming", *Artificial Intelligence*, vol. 72, no. 1-2, pp. 81-138, 1995. Available: 10.1016/0004-3702(94)00011-o.

[27] A. Barto, "Reinforcement Learning and Dynamic Programming", *IFAC Proceedings Volumes*, vol. 28, no. 15, pp. 407-412, 1995. Available: 10.1016/s1474-6670(17)45266-9.

[28] Jing Peng and R. Williams, "Efficient Learning and Planning Within the Dyna Framework", *Adaptive Behavior*, vol. 1, no. 4, pp. 437-454, 1993. Available: 10.1177/105971239300100403.

[29] P. Tadepalli and D. Ok, "Model-based average reward reinforcement learning", *Artificial Intelligence*, vol. 100, no. 1-2, pp. 177-224, 1998. Available: 10.1016/s0004-3702(98)00002-2.

[30] "Revenue management", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Revenue_management. [Accessed: 07- Jan- 2019].

[31] D. Bertsekas, J. Tsitsiklis and C. Wu, "Reinforcement Learning in POMDP's via Direct Gradient Ascent", *Journal of Heuristics*, vol. 3, no. 3, pp. 245-262, 1997. Available: 10.1023/a:1009635226865.

[32] B. Van Roy et al., "A Neuro-Dynamic Programming Approach to Retailer Inventory Management", 1997. [Accessed 7 January 2019].